CrayViz, a Tool for Visualizing Job Status and Routing in 3D on the Cray XT3

John Biddiscombe, *and* **Neil Stringfellow**, *Swiss National Supercomputing Centre*

ABSTRACT: *When initially installed, the Cray XT3 at CSCS showed node failures which appeared to occur in a pattern. The tools available for the analysis of the failures were limited to 2D printouts of node status and were difficult to interpret given the 3 dimensional connectivity between the processors. A visualization tool capable of displaying the status of compute nodes and the routes between them was developed to assist in the diagnosis of the failures. It has the potential to assist in the analysis of the working system and the development of improved allocation and job scheduling algorithms..*

KEYWORDS: Cray XT3, Visualization, Network Topology, Tool

1. Introduction

A pattern of node failures which occurred on the Cray XT3 systems shortly after their introduction in the early summer of 2005, led CSCS to request the help of the visualization group in developing a tool to analyse routes taken by network traffic when traversing the 3-D Torus topology of the machine, since translating 2-D textual representations of the physical layouts of the cabinets into probable routing information was proving difficult with the tools which were then available.

Although the node failure mechanism was identified, and a solution provided before the tool was ready to be used, the overall ability to visualize the status of the machine had shown great potential and therefore this utility was developed further to analyse job placement and node failure information in a historical context.

With the continuing maturation of the Cray XT3, and as more performance monitoring libraries are ported to this architecture, there is a place for. further development of the tool to carry out enhanced performance monitoring and network contention analyses.

Historical Context

The early Cray XT3 systems, which were delivered in late 2004 and the first half of 2005, had to be rebooted several times per day due to large numbers of unexplained node failures, and the frequency of these node failures meant that it was difficult to carry out useful scientific research in the intervals between reboots. This limitation was enhanced by the fact that the nature of the node failures appeared to spread rapidly around the machine as though there were a contagion on the interconnect, and furthermore the in-built RAS system which attempted to identify failed nodes was sometimes unable to detect nodes which were unusable.

In order to get a better picture of what was happening on the system at CSCS, an approach was made Pittsburgh Supercomputer Centre (PSC) who had a utility to query the operational status of several nodes. The base `ping_node` tool to ping a single node had been written by Cray and Sandia National Laboratories and this was then further developed by PSC [**1**] to a `ping_list` utility in order to allow it to ping several nodes at once. The original single node utility was deemed unsuitable for studying the overall state of the machine as each call could take several seconds per node, and on a system consisting of over a thousand processors there would be a strong possibility that a system reboot would be required while collecting the information.

Using this tool to enhance the standard machine viewing tools allowed the failures to be "seen" on the machine, and it was soon clear that the node failures occurred in regular patterns with either the same positioned node on several cabinets failing, or else the same nodes in a vertical or horizontal line on a cabinet appearing as failures - it should be noted that other sites with different classes of topology may have seen different failure patterns.

By using different nodes to carry out the ping, different failure patterns could be seen, implying that not all of the nodes had really failed, and furthermore the output from the `ping_list` utility could be combined with the information held in the central database to show nodes which were not able to be pinged from a login node, but which the system still thought were alive and would therefore be considered as eligible for a job to be launched on them.

A basic knowledge of the interconnect topology and the simple routing algorithms permitted an insight into what was happening - only one or two nodes were failing, but it was the SeaStar which was having the problem and this disruption to the high speed network stopped all traffic through the SeaStar. This effectively meant that any nodes which could only be pinged from a certain position by traffic passing through the node on either the outward or inward direction would become invisible from the pinging node. Although the `ping_list` utility was invaluable in this analysis, the text-based tools for collecting this information were insufficient for gaining true insight into the state of the system.

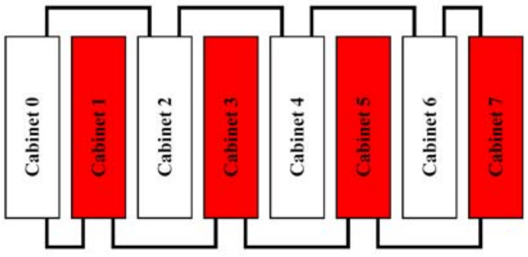


Figure 1: Network connections for an 8 cabinet system

Physical and Logical Layouts

The arrangement of nodes, blades, chassis and cabinets in physical terms follows simple rules to build up multiple rows of a system from basic building blocks, but the connection between the nodes is dependent upon the number of cabinets in a configuration which determines the topology of the 3-D Torus. There are 4 classes of topology, ranging from a class 0 Topology which can have up to 3 cabinets, then there is a class 1 topology which consists of a single row of cabinets and through to class 2 and 3 topologies which consist of 2 or more rows of cabinets. The class of topology is important in determining which chassis and cabinets are connected to each other and the physical distance between two cabinets does not determine how many SeaStar routing chips a message has to pass through when communicating between them. To determine the route which a message has to take between two nodes, it is necessary to consider the logical layout of the processors in the 3-D Torus.

Apart from at the ends of each row, adjacent physical cabinets are not normally directly connected to each other by cables for any topology which is greater than class 0. For example, in an 8 cabinet class 1 topology the cabinets are connected as in Figure 1 where adjacent cabinets in the logical layout are as shown in routing (1)

$$0 \leftrightarrow 2 \leftrightarrow 4 \leftrightarrow 6 \leftrightarrow 7 \leftrightarrow 5 \leftrightarrow 3 \leftrightarrow 1 \leftrightarrow 0 \quad (1)$$

This type of connection scheme minimises the cable length as joining together adjacent cabinets would require a very long cable to connect the end cabinets.

From this method of connecting chassis and cabinets together in continuous loops, logically all topologies are actually 3-D Torus configurations so that two routes can always be taken between two points in a given dimension. These toroidal configurations can be thought of as wraparound cubes – analogies include computer games such as Asteroids and Pac-Man where the screen is a square, but when an object disappears from one side of the screen it reappears on the other side.

For the purpose of analysing the routes between any two points, it can be seen that the physical location of cabinets is unimportant but that the logical connections in the 3-D Torus are the significant element, and from this information it is then also necessary to know what route would be taken by a message between 2 points. On the Cray XT3 static routes are pre-calculated at machine boot time for how to get from one node to another, and this information is then dispatched to the SeaStar processors to use for all routing of messages.

In most cases the forward and return routes between two points are different, and only for nodes which have 2 of the X, Y and Z coordinates the same do the forward and return routes coincide. As an example, assuming that there was no wraparound and that the topology was a simple mesh then traversing between logical points (0,3,3) and (3,2,4) would involve passing through all of the intermediate nodes along the routes shown in routes (2) and (3)

$$\begin{aligned}(0,3,3) &\Rightarrow (0,3,4) \Rightarrow (0,2,4) \Rightarrow (1,2,4)\\ &\Rightarrow (2,2,4) \Rightarrow (3,2,4)\end{aligned} \quad (2)$$

$$\begin{aligned}(3,2,4) &\Rightarrow (3,2,3) \Rightarrow (3,3,3) \Rightarrow (2,3,3)\\ &\Rightarrow (1,3,3) \Rightarrow (0,3,3)\end{aligned} \quad (3)$$

In its simplest form, the traversal of different routes on the 3-D Torus involves a preference for moving along the network in the priority order Z+,Z-,Y+,Y-,X+,X- and this means that in our 8 cabinet example shown in figure [1] that a ping between a node on cabinet 0 and cabinet 7 would move through the cabinets in the order $0 \Rightarrow 2 \Rightarrow 4 \Rightarrow 6 \Rightarrow 7$ on the outward direction and then $7 \Rightarrow 5 \Rightarrow 3 \Rightarrow 1 \Rightarrow 0$ in the return direction. This means that a ping-pong message between any two nodes which were separated by half of the machine in one of the dimensions would cross all of the planes in that dimension on either the outward or return journeys.

Despite the fact that the logical depiction of the 3-D Torus topology is the correct method to determine a metric of how close two nodes are on the machine and therefore the shortest path between two nodes, for historical reasons the processors are given out to jobs in a very simple manner from the first processor number in the database onwards, and this means that the nodes are given out in terms of physical cabinets instead of the best approach for minimising communication along the high speed network. The short hop time between SeaStars means that this is not so significant for latency, but bandwidth contention between many messages trying to traverse the machine could potentially have a high impact on the performance of communication intensive applications. For example, in the 8 cabinet example nodes for a job might be allocated on the physically adjacent cabinets numbered 3 and 4 even though they are on opposite sides of the machine as far as the high speed interconnect is concerned.

Text Based Tools

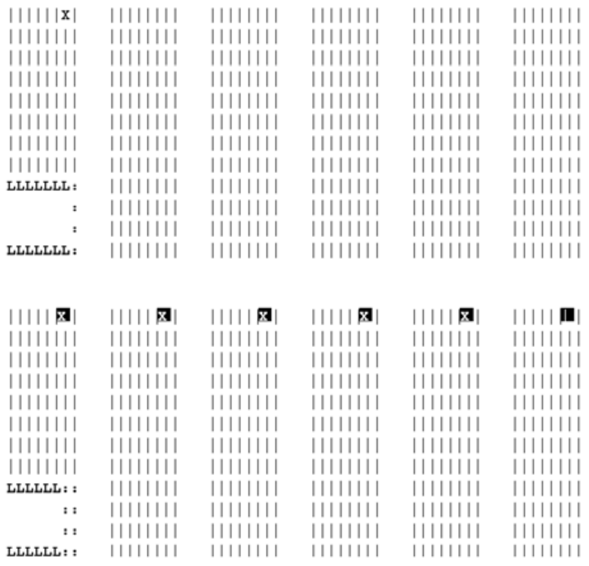
The original text-based tools for viewing the operation of the machine were very soon shown to be inadequate for carrying out any sort of analysis of the failure patterns, and enhancements to the standard utilities were therefore required. As well as the invaluable `ping_list` utility, the main additional tool available was `xtshowmesh` which gave an overview of node utilisation with information on which nodes currently had jobs running on them which nodes were free for batch and interactive use, and most importantly for the analysis, which nodes were marked as down in the system database.

The original version of `xtshowmesh` provided its information in a format which was more suited to the Red Storm architecture than the XT3 in that the cabinets were displayed as 24 columns each with the 4 processors on a blade, and the traversal of the columns was from blade 0 to blade 7 along the bottom row, then from blade 7 to blade 0 in reverse on the second row and finally from blade 0 to blade 7 on the top row. This format made it very difficult to read when considering the layout of the 3-D Torus topology on the Cray XT3 and therefore this output was rearranged and tied together with the results of carrying out `ping_list` on a cabinet by cabinet basis to give a newer utility for studying the status of the machine.

A sample output of the new utility is shown in Figure 2 where the cabinets are now arranged in two rows such that a message passes between cabinets in a circular direction either clockwise or anti-clockwise depending upon the shortest distance between them. Failed nodes from the database are shown as an ``X" and failed nodes from `ping_list` are shown in reverse video, with failed nodes in both the database and `ping_list` being shown as a reverse video 'X'.

This example shows that the cabinets along the bottom row (in this case the even numbered cabinets) all show a failure from `ping_list`, but the final cabinet on the bottom row has no failures according to the database, while the first cabinet on the top row does have a failure according to the database.

The layout of machine information in cabinets was later implemented in Cray's `xtshowcabs` utility, as this is a much more useful way of seeing the system on a Cray XT3 as opposed to viewing the status of Red Storm. Although the combined tool allowed a better viewing of the system it was still inadequate for a true analysis of the problem as routes between nodes still required a translation from a 2-D representation into a 3-D perception in the mind of the analyst.



```
||||||X|  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
LLLLLLL:  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
       :  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
       :  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
LLLLLLL:  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||

|||||X||  |||||X||  |||||X||  |||||X||  |||||X||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
||||||||  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
LLLLLL::  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
      ::  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
      ::  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
LLLLLL::  ||||||||  ||||||||  ||||||||  ||||||||  ||||||||
```

Figure 2: Sample output showing `xtshowmesh` and `ping_list` combined

A Graphical Visualization Tool

In order to further analyse the problem on the Cray XT3, CSCS was able to make use of the fact that it already has a well established scientific visualization group providing services to the Swiss scientific research community. It was clear that the skills of this group could be utilised to transform the physical representation of cabinets in 2-D into a logical representation of the network topology in 3-D.

With a vague specification of what was required, a visualization tool was rapidly developed which allowed either a representation of the machine in terms of the layout of physical cabinets, or allowed a 3-D visualization of the logical layout according to the network topology so that it was easily possible to see which nodes were really next to each other on the 3-D Torus.

By analysing the routing information which was output at boot time on the system management workstation (SMW), the routing algorithm could be deduced and this allowed the tool to be enhanced so that it was possible to visualize the real paths through the machine which would be taken by messages between two nodes.

The visualization tool incorporated some of the text based tools for information, and was able to use `ping_list` from different nodes to have alternative views of which nodes had failed, although due to the agglomeration of service nodes at the bottom of only 2 cabinets, the routes taken when issuing `ping_list` where not drastically altered by using a different login node. An early version of the tool can be seen in figure Figure 3 which shows each node in the machine as a point on a grid. Nodes which have failed are drawn as large coloured cubes and the route from the login (ping) node to the failed node is shown. Forward and return paths are shown using different colours – they are mirror images of each other when viewed singly.

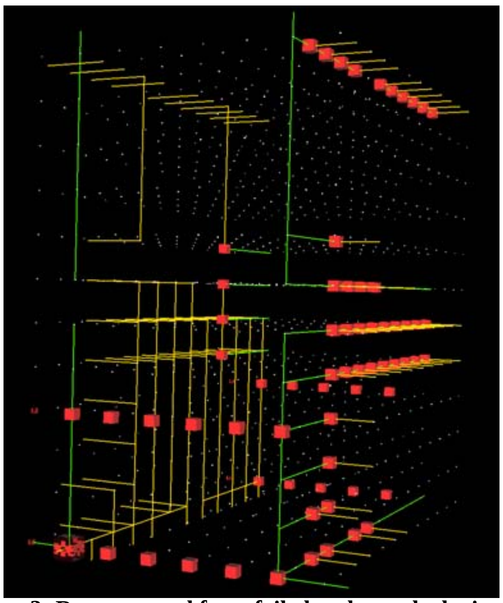


Figure 3: Routes to and from failed nodes to the login node

Just as the visualization tool was becoming useful for this analysis and it was becoming possible to identify the nodes which were the probable culprits for multiple node failures, Cray engineers carried out system maintenance on the SeaStar processors where a small voltage increase from 1.5 to 1.6 volts made the problem disappear completely. Fortunately, the effort to develop the tool was not wasted as new possibilities were evident in the areas of gaining a system overview of running jobs, historical information about node usage and the potential to analyse how job performance could be affected by network bandwidth contention.

Later Tool Development

It was clear that this tool was still very useful for gaining information about the usage of the system at the time at which the tool was running, and this was very clear when seeing how jobs were distributed around the 3-D Torus. Some jobs appeared to be allocated in a manner which meant that there were short distances between nodes when looking at the output from the text based tools showing allocations by cabinet, but using the visualization tool it could now be seen that they were in fact poorly distributed across the 3-D Torus and that jobs running on other nodes which were themselves poorly distributed would cause the potential for bandwidth contention.

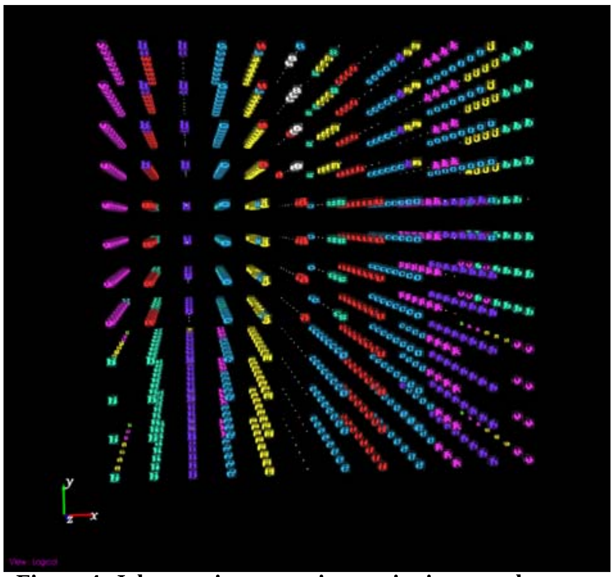


Figure 4: Jobs running at one instant in time are shown as coloured cubes (the JobCube), allowing their distribution on the machine to be visualized immediately

An investigation of the information held in the MySQL accounting database revealed that it contained all of the necessary information about jobs, the nodes which were allocated to jobs and node failure information, such that the tool could be used for historical data as well as taking a snapshot of the running machine (see Figure 4). The only aspect which would then be missing from the design of the original tool would be the ability to ping nodes, but with the voltage increase on the SeaStar processors there were no known occasions where node failures in the database were different from information gained from `ping_list`.

For this reason the tool was then changed to directly query the database for information about jobs, and it was now possible to take snapshots of the state of the machine at various times, to look for possible node failure patterns in a given period, and to have dynamically changing information in the form of time-line animations, either between fixed points in time or analyses such as the changing state of the machine during the lifetime of a particular job.

A further enhancement came when Cray put the correct 3-D topology grid coordinates for an XT3 system in the MySQL administration database instead of the coordinates for a Red Storm system, and this had the side-effect that `xtshowmesh` no longer worked on the XT3 and the `xtshowcabs` utility was good enough for general usage whereas the visualization tool itself was the only real way to write an `xtshowtorus` utility. The ability to take this information from the database meant that the tool was now fully portable and could be taken to any other Cray XT3 and used without any extra effort. Unfortunately the routing information was still not held on the database, and therefore the routing algorithm still needed to be hard coded into the tool. However a problem with hard-coding this information was that a failure to initiate nodes at boot time would lead to a different routing algorithm from the default being employed in the network but there was no way for the tool to know this.

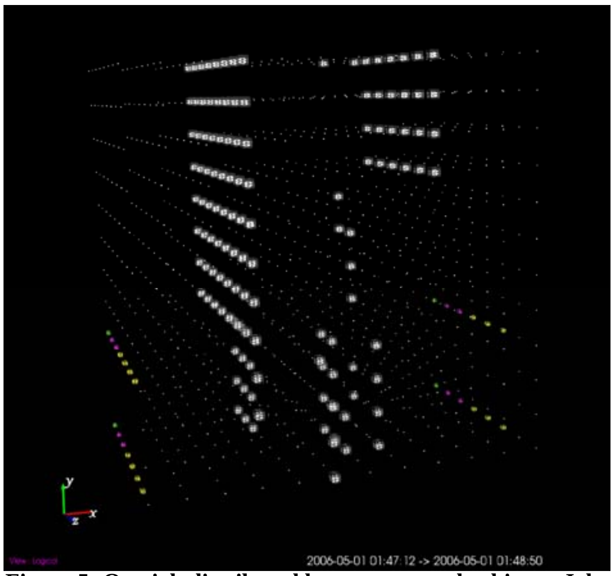


Figure 5: One job distributed between several cabinets. Jobs running on the in-between nodes will cause traffic congestion.

Then a further development came with the inclusion of an option to carry out analysis of theoretical link contention based on expected communication patterns, with three main options being available to an analyst:-

All-to-all

It is assumed that random access between processes would occur and therefore all routes are calculated in order to determine likely network contention.

Nearest neighbour

A simple communication pattern of sending messages to the nearest neighbour with a 1-D wraparound is used to determine routes. Since some distributions of processors are sufficiently separated even this type of communication could cause bandwidth contention.

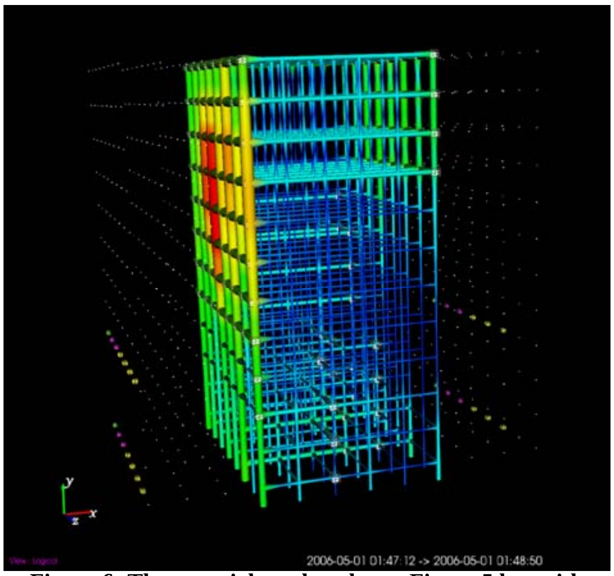


Figure 6: The same job as that shown Figure 5 but with routing of each node to each other, the region where traffic is densest appears red

Square and Cubic Mesh neighbours

The dimensions of a 2-D or 3-D grid could be entered so that communication in this manner would be calculated as this type of messaging is typical in many high performance computing applications.

An example of an analysis of network contention caused by a job is shown in Figure 6

Additional output from the tool

Once the routing analysis has been computed it is possible to produce some statistics, such as the mean and standard deviation of route lengths from one node to another for a particular job. Should the same job be run on different occasions with significantly better or worse performance during one run, such information could give clues as to the reason for the difference.

A test for this routing analysis capability was provided when a job which had been submitted more recently was found to have computed only 30% as much data as the same job submitted some months before. Looking at the distribution of compute nodes and the traffic patterns which would arise from the two jobs made it clear that the earlier job should run more efficiently. However, no direct link between the route lengths or traffic patterns could ever be made without detailed data collection and examination for both runs – and it seems unlikely that such a large performance reduction could have been caused by traffic contention alone.

CrayViz is now useful for examining historical information from a theoretical standpoint, but the real value of this tool could come with an upgrade from theoretical routing within an application to the actual communication patterns which occurred during a run. Although the collection of all message information such as start and finish times, source and destination and amount of data would be cumbersome, general information patterns could be used and in this context the IPM tool [2], which is currently being ported to the Cray XT3 would be a good candidate for general performance data collection.

It may also be possible to directly gain real data collected from the SeaStar processors, whether this be actual runtime collection of data for real-time analysis, or a summary data collection at the end of a job run which would be more suitable for off-line analysis.

Although there are now few occasions where failures in SeaStars occur, and therefore the simple routing algorithm used by CrayViz could be expected to correctly calculate the paths between processors, the current hard-coding of this routing algorithm into the program does not fit well with the elegance of the rest of the code which uses the MySQL database wherever possible allowing the visualization to be based entirely on external information. Dumping the routing information onto the database at boot time with a time stamp to indicate the start period for which this routing information is valid, would be a much better way of allowing a full historical analysis of job, performance and network contention.

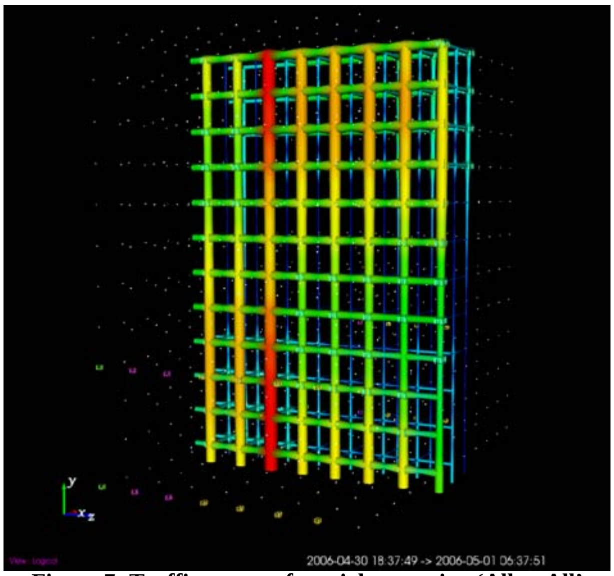


Figure 7: Traffic pattern for a job assuming 'All to All' communication between nodes

Correct calculation of routing could be used to avoid potential bottlenecks when either multiple jobs have hot-spots in the same region of the machine, or when a failed node causes an unusually large amount of traffic to pass through a certain region. Consider Figure 7 and Figure 8 which show theoretical routing for the same job assuming an 'All to All' traffic pattern in one and a 2d Grid computation with a 4x4 size in the other. The hot spots in terms of traffic are in quite different locations. If another job is allocated in such a way that it also places heavy network demands in the same region – or if a failed node causes rerouting of neighbouring traffic through this region, then the performance of the job may be unpredictable.

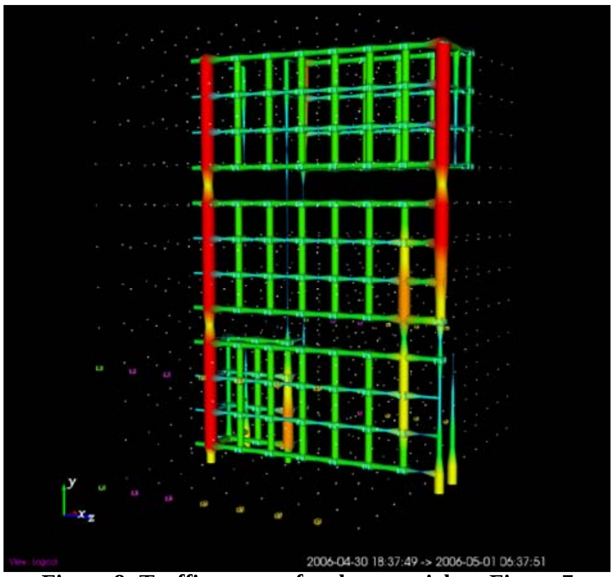


Figure 8: Traffic pattern for the same job as Figure 7 assuming a traffic pattern of a 2D 4x4 grid

Conclusions

The CrayViz tool was originally developed to help identify patterns produced by a node failure mechanism, but the reason for the failures was identified and a solution put in place before the tool had been able to be used for the pattern analysis.

The development of a 3-D visualization tool for studying the Cray has proved to be very valuable in understanding processor distribution within jobs, node failures and possible network contention based on theoretical communication patterns. The incorporation of the correct network topology information in the MySQL database allowed CrayViz to become fully portable between XT3 machines.

Future development of tools such as IPM to gather information from applications on real communication patterns would enhance the potential of CrayViz to be used as a tool for bandwidth contention analysis.

Obtaining summary information from the SeaStar processors could also be useful, particularly in determining any build up of message queues to indicate bandwidth contention.

About the Authors

John A. Biddiscombe completed his degree in electronic engineering at Warwick University in 1989. From 1990 onwards he held several positions at the Rutherford Appleton Laboratory, in 1995 he moved to the Radio Communications Research Unit where he became software development manager for theoretical modelling, working on radio propagation and 3D simulation tools. In 2004 he joined the visualization group of CSCS bringing his specialized VTK knowledge to the group. In this role he has worked on a variety of visualization and programming projects for the CSCS user community.

Neil D. Stringfellow studied Mathematics at undergraduate and master's level before attending Cranfield University where he obtaining a PhD also in Mathematics. After completing his post-doctoral studies he joined the staff of the University of Manchester where he provided application and optimisation support to scientific researchers who used CSAR, one of the United Kingdom's national supercomputing services. Neil joined the staff of CSCS in 2004 and since then he has been providing support to specific groups from CSCS' user community and he has also been involved in the introduction and development of the Cray XT3 supercomputer.

References

[1] John Kochmar Chad Vizino, Nathan Stone and J. Ray Scott. Batch scheduling on the cray xt3. In Cray User Group 2005, Alberquerque, New Mexico, May 16-19, 2005, 2005. available at http://www.cug.org/5-members only/1-attendees/proceedings attendee lists/2005CD/S05 Proceedings/pages/Authors/Vizino/Vizino paper.pdf (cited May 2006).

[2] David Skinner of NERSC. Integrated performance monitoring. Available at (cited May 2006) http://sourceforge.net/projects/ipm-hpc